Design of a Hypermedia Interface Translating between Associative and Formal Representations

Francis HEYLIGHEN*

PESP, Free University of Brussels, Pleinlaan 2, B-1050 Brussels, Belgium

ABSTRACT. It is argued that in order to efficiently tackle complex problems, user and support system should intimately interact, complementing each other's weaknesses. Strengths and limitations of human intelligence, respectively computer intelligence, can be derived from the mechanism of associative, respectively "chunk-based", memory. A good interactive interface should hence allow one to translate between associative (context-dependent) and chunk-based (formal) representations. Associative knowledge can be expressed more explicitly through hypermedia, consisting of a network of connected chunks. Different mechanisms for supporting the creation of networks are reviewed: check-lists, outlining, graphic representations, search functions, ... These mechanisms can be complemented by looking for "closed" subnetworks, which define invariant, formal constraints, which can be used to guide inferences. A prototype implementation of an interactive interface, the CONCEPTORGANIZER, is sketched, and some potential applications in the areas of idea processing, knowledge elicitation, decision support, and CSCW are outlined.

1. Introduction: open and closed support systems

Many systems have already been implemented on computers with the aim of supporting the user in problem-solving. Such support systems can be roughly divided in two classes: *open* and *closed* systems. A closed support system may be viewed as a system substituting for the user: the system "thinks" instead of the user, the user is limited to the passive role of introducing the data defining the problem and waiting for the system to find a solution. With an open support system, on the other hand, it is the user who uses his or her intelligence and creativity in order to solve the problem, the system merely functions as a medium which makes it easier for the user to express his or her ideas.

Examples of closed systems are traditional algorithmic programs, e.g. for computing the solutions to an equation, or to a less degree, traditional expert systems, where the system replaces the human expert, and where the activities of the user are limited to questioning the system, and providing the needed data. Examples of open systems are applications available on most personal computers,

* Senior Research Assistant NFWO (Belgian National Fund for Scientific Research)

such as word processors, graphic packages, outliners... Here the burden of writing the text, designing the graph, planning the project..., is completely on the user. The system just makes the task a little easier by providing tools that allow the user to represent and manipulate his or her ideas in a more transparent and effective way.

The advantage of an open system is that it is very flexible: it imposes few restrictions on the type of problem which is to be solved or on the format in which the data are to be introduced. A closed system, on the other hand, has a very specific range of application or of expertise. For example, an expert system may only support the solution of problems in a specialized domain of medicine, e.g. audiological diseases. An outliner, on the other hand, can be used for the design of solutions to problems in almost any domain. The drawback of an open system is that the amount of support or help the system provides is in general much smaller than that of a closed system. For example, an outliner may be used by a doctor for jotting down and classifying the observations on a particular patient. This outline can then help to structure the doctor's thoughts when making a diagnosis. Yet the outliner itself cannot tell how to cure the disease, although an expert system might. An open system appears less “knowledgeable” or “expert”, perhaps even less “intelligent”, than a closed system. There seems to be a *trade-off* between expertise and flexibility: if one moves from the more “open” end of the spectrum of support systems, to the more “closed” end, one will gain in expertise but lose in flexibility.

Is it then impossible to combine the advantages of both approaches? Let us examine the human-computer interaction. Neither of the systems sketched above is characterized by a real interaction: in a closed system the user cannot intervene in the problem-solving process once the problem has been defined, he or she must wait for the result; in an open system, the system cannot redirect the way the user tackles his or her problem, it just provides tools that the user may or may not utilize. In a truly interactive approach, user and system would both be active. They would work on the problem in parallel, and continuously exchange data and provisional results, thus complementing each other's activities. Neither of them would be in complete control of the process, though the final decision about which problem to solve or which solution to prefer would of course remain with the user.

A possible paradigm for conceptualizing such an “intimate” man-machine interaction (cf. Beyls, 1986) is provided by Pask's concept of *conversation* (Pask, 1980; Pask & Gregory, 1986). A conversation is a symmetric interaction between two actors in which new concepts are constructed by comparing, combining and refining the provisional concepts introduced by the individual actors, until an “agreement” is reached about a shared meaning. In the present case, the user might for example introduce some provisional concepts describing aspects of the problem, and the system would then analyse these concepts, perhaps add data from its knowledge store, and propose a possibly more adequate reformulation to

the user. The user would then either accept this newly formulated system of concepts, or change it by adding new insights or correcting inappropriate assumptions made by the system. The system could then again react on this changed concept, and so this interactive reformulation of concepts would proceed, until user and system would "agree" about an adequate representation of the problem which would be best suited for selecting the final solution.

This "conversational" metaphor for man-machine interaction is stimulating, but it should not mislead us to consider human and computer as essentially equivalent or similar partners in the problem-solving process. There is little advantage in having a computer mimic the conversation of another human actor if one can have the real thing. The benefit of a man-machine partnership should just reside in *complementarity*, i.e. in letting the machine do what is difficult for the human actor, and vice versa. In that way, man and machine working in "symbiosis" could solve more problems than the sum of the problems solved by either of them in separation. In order to design such a system which could effectively complement the human user in the interaction, we must first analyse the specific strengths and weaknesses of human problem-solving capability.

2. Strengths and limitations of human memory and intelligence

2.1. HUMAN VS. COMPUTER MEMORY

It is a well-known fact that computer memory functions in way very different from human memory. In a computer (at least on the machine level), data are stored in discrete, disconnected *chunks*, the number of which is virtually unlimited. Data are retrieved by a sequential search through all stored information until the appropriate chunk is found. This method guarantees: a) that what is "recalled" is identical to what was stored; b) that what was stored will be effectively retrieved, in a mean time proportional to the amount of data stored.

In humans, two mechanisms can be distinguished: *short term memory* and *long term memory* (Wickelgren, 1977). Short term memory is most similar to computer memory: it makes it possible to store chunks of information, and to recall them relatively directly. However, the storage capacity is very limited compared to computer memory: according to Miller's (1956) classical results only 7 ± 2 chunks may be stored simultaneously (more recent findings appear even more pessimistic, cf. Warfield, 1988). Since the period during which this information remains stored is by definition short, there is moreover no guarantee that a particular chunk will still be in memory when it is needed.

Long term memory has virtually unlimited storage capacity (though the term "storage" is here hardly appropriate). There are two basic retrieval mechanisms: recall and recognition. Recall is comparable to the retrieval of chunks sat-

isfying specific conditions. However, recall is in general a very unreliable mechanism. There is no guarantee: a) that what is recalled is identical to what was stored (cf. biased reconstructions of observations, as exemplified by many stories of witnesses, Wickelgren, 1977); b) that anything stored may be recalled at all (cf. phenomena such as forgetting, “it's on the tip of my tongue”, and repression).

Recognition, on the other hand, is a very fast and flexible mechanism. Instead of retrieving specific data in the form of chunks, it activates response patterns by matching (perceptual) input patterns. This activation is, however, *selective*: input patterns for which there is no matching internal pattern are hardly noticed (they are not “recognized”), and internal patterns which are not activated by the appropriate input pattern may remain buried forever in long term memory. This mechanism is basically *associative*: the activation “spreads” from one pattern to another pattern which is connected to it by associations. These associative connections are gradually built up through learning by experience, as exemplified by classical conditioning through reinforcement (Wickelgren, 1977).

The spread of the activation can be compared with the flow of a river through a landscape in which valleys and channels are eroded (cf. de Bono, 1969). The more water (activation) flows through a river, the deeper the valley will be eroded (the stronger the association becomes), and hence the more difficult it becomes for the water (activation) to choose an alternative path. This explains the “conservative” bias in associative thinking: the more a particular association has been activated in the past, the more likely that it will be activated in the future, and the more difficult it will be for an alternative association to get activated. This mechanism has a strong advantage if recurring regularities are to be learned, but it is a genuine drawback if a problem requires a new way of looking at things—i.e. creativity (de Bono, 1969).

A similar associative learning mechanism is now being implemented on computers with the help of the “neural network” paradigm (cf. Rumelhart & McClelland, 1986). However, it seems hardly likely that such “connectionist” systems will reach the possibilities of human memory in the near future: first, the “hardware” capacity of the human brain (number of neurons and number of synapses connecting neurons) is so enormous that it will take us many more years, or rather decades, before anything comparable could be built by means of microprocessors; second, and perhaps more important, the “software” contained in the brain of a typical adult human being, i.e. the whole of explicit and implicit knowledge, has been built up during a lifetime of experience, in which the human actor was confronted with the most diverse situations, which were perceived and acted upon by means of the most sensitive and intricate sensory-motor systems. Apart from experimental situations, it seems rather pointless to try to duplicate the complexity of human experience, by for example building an autonomous, connectionist robot (cf. Maurer & Schreter, 1990) that would learn the same way a human child would learn. Though such a robot would be a good

instrument for testing scientific hypotheses, like we remarked before there is no point in manufacturing artificial human beings (as distinguished from "expert" AI or connectionist systems) if it is much cheaper to get the real ones.

Let us summarize the difference between human and computer memory:

Computer memory is

- *discrete* (digital),
- *invariant*, and hence
- *reliably retrieved*.

Long-term memory is

- *continuous*, organized as a network of *associations*,
- which *changes* continuously through experience (or simply by using it)
- *not reliable* when retrieval or exact reproduction is required, since only those aspects will be retrieved which associatively "match" the other patterns.

Long term memory is flexible and robust under slowly changing circumstances, but inflexible when something completely new must be grasped. Short term memory can be used to bridge the gap between the disconnected (chunks) and the connected (associations), but has a very limited capacity.

2.2. BOUNDED RATIONALITY

What are the consequences of this specific memory organization on problem-solving? The very dense and extended network of associations makes it possible to rapidly recognize aspects of situations which have been experienced before, activating the corresponding response patterns, and to (slowly) learn aspects which are new. However, when many aspects of the problem are new, and there is little time to learn, long term memory becomes virtually useless. It may even become counterproductive, when the action patterns which are activated are inappropriate for the problem at hand. In that case it would be better if recognition by association could be (temporarily) turned off, and replaced by a more systematic, less biased exploration of possible actions.

Short term memory may partially compensate for this lack of flexibility, if the problem can be reduced to a few discrete chunks. In that case, new combinations of the chunks (representing possible actions towards the solution of the problem) can be hold in short term memory and tested as to their potential for solving the problem. (This mechanism enabling recombination of chunks or "concepts" forms the basis of the rational-conceptual level of cognition, which distinguishes humans from animals, cf. Heylighen, 1990a). It is clear that the 7 ± 2 boundary severely limits the number of possible combinations which can be explored in this way. (Warfield (1988) considers the combinations as the power set of the set of the initial number of aspects to be considered, and thus concludes that this initial number cannot be greater than 3 ± 0.)

The consequence of this restriction may be called *bounded rationality* (Simon, 1957). Perfect or unbounded rationality might be defined as the systematic search through *all* possible combinations of features defining a problem space, in order to find the one which is *optimal* with respect to the evaluation criterion defining the problem. Bounded rationality is characterized by a *partial* exploration of the search space until a *satisfying* solution is found. It is well-known that computers too are bounded in their capacity for exploring problem spaces, but their limits evidently lie much farther away than the one determined by the magical number 7.

2.3. HUMAN-COMPUTER COMPLEMENTARITY

The difficulty with conventional computers is that they require stable chunks of information to work with, in other words the problem domain must have been split up into *distinct* parts, such that the distinctions are *invariant*. Once the distinctions between chunks are fixed, the computer can manipulate the chunks in whatever way is needed. However, if no distinctions are given beforehand, i.e. if the problem is not *well-structured*, the computer is helpless, because it cannot distinguish the relevant features of a problem on its own. In practice, a computer program will need a human user to introduce the data in the form of discrete chunks.

Though the cognitive system of a human is capable of distinction, these distinctions are in general not invariant (cf. Heylighen, 1990d). They depend on the state of the whole associative network, which is determined by all the things experienced, either through perception, through thinking or through action. In other words, the distinctions made by a human observer, which define the content or meaning of the chunks he or she has in mind, depend on the *context*, i.e. on phenomena and events outside the strict problem formulation. If one wants to represent the problem in a way that can be grasped by the computer, one will have to *formalize* one's awareness of the problem. “Formal” can generally be defined as “having an *invariant* meaning”, i.e. the meaning and hence the distinction does not change when the context changes, it is *context-independent*.

It would then seem sufficient to support the formalization of the domain by the user, transforming an ill-defined problem into a well-structured one. However, most real-world problems are *complex* (Heylighen, 1988, 1989a) or *externally structured* (de Zeeuw, 1989): this means that the factors on which the problem depends cannot be completely formalized, because they are changeful, vague or ambiguous; there will always remain some factor external to the problem representation influencing the chances—and even the criteria—of success; the context cannot be eliminated. This means that any provisional formalization of the problem will have to be reinterpreted in the light of new events, new data or new associations which are triggered by it, possibly leading to yet another (partially) formal model. In this way we are led to a “conversational” dynamics, where problem (re)formulations and tentative solutions, made up of

distinctions and associations, are continually exchanged between the discrete memory of a computer and the associative memory of a human user, until a satisfying solution is "agreed upon".

A support system designed to stimulate this type of interaction will require an interface able to translate discrete-formal and associative, context-dependent representations of a problem in the most direct way possible. We shall now examine some principles for the design of such an interface.

3. Expressing associative knowledge through hypermedia

3.1. EVOCATION OF ASSOCIATIONS

The first step in the formal-associative conversion consists in supporting the expression of associative ideas in a format which can be stored by the computer, but with a minimum of artificial restrictions for the human. When transferring an idea to computer memory, we want to maintain as much as possible of the richness of associations it originally had in human memory.

Sensory-motor interface

Therefore one must remember that associations are primarily evoked or triggered by sensori-motor stimuli. This means that the more sensory dimensions are involved, the more associations will be evoked. The most important sensory modalities are vision, hearing and muscular (proprioceptive) sensation. This means that our computer interface should integrate several interaction media, containing at least a graphical display (preferably high-resolution, colour, and capable of animation), an audio output (preferably high quality stereo), and a device for direct muscular manipulation—e.g. a "mouse". Since these facilities are already available on many personal computers, this should not pose any hardware problems.

On the software level, however, the problem will consist in effectively integrating these media in the system, so that they are used in practice by the user for expressing his or her ideas, instead of just being there as "gimmicks". For example, a practical use for colour is to allow the user to associate entities (chunks, links, commands, ...) of different types with different colours, in such a way that related entities have similar colours. This makes it much easier for the user to recognize the things he or she is looking for in a complex visual display containing many objects or windows (Begeman & Conklin, 1988). Similarly, different types of events occuring in the system (for example, finding a string) could be announced by different sounds.

Virtual reality

In a later stage, one may conceive of even more direct sensory-motor interaction channels. For example, there already exists a commercial head-mounted display,

called "EyePhone™", offering 3-dimensional colour vision, stereo headphone, microphone, and even facial expression sensors. This device is to be complemented by a DataGlove, enabling intricate 3-dimensional manipulations of the "virtual objects" that are seen on the display (Williams, 1988). This makes it possible to reconstruct a complete *virtual reality* (also called *cyberspace*) inside the computer, such that the user has the impression that he or she senses and manipulates the virtual objects as if they were real (Ditlea, 1989). (One might even speculate that in a later future direct connections between brain activities and processes inside a computer would be possible, for example by means of a sophisticated device for registering and inducing brain waves. In that case, the virtual reality of the problem representation and its objects would be generated directly, in the same way as a dream, i.e. without any intervening sensory stimuli.)

Natural language

Another essential medium for evoking associations, which is not directly sensory-motor, is (natural or verbal) *language*. Language in everyday reality is used mostly in an informal way: the meanings of the words and sentences are not fixed by definition, but depend on the *context*, which includes the conversational interaction, the overall setting, the things said before, metaphors and analogies.

The associative character of natural language can be illustrated by the numerous little phrases used in texts, such as this one, for explanation: "*One must remember that…*", "*On the other hand, we have...*", "*As mentioned before, ...*", "*This may be illustrated by...*", "*This leads us to consider...*". On the formal level, these phrases are simply redundant, they do not convey any information. A purely formal text would consist of disconnected propositions, such as: *B(c); for all x, A(x) implies B(x); there exists a y, such that C(y, t)* ... Clearly it would be very difficult to understand such a text without any phrases helping to make the appropriate associations between what the sentence tries to convey, and the context. The context consists here of all the things which are already in associative memory, either because they have been read before in the same text, or because they belong to a general store of knowledge the author assumes he or she shares with the reader. One may conclude that our man-machine interface should be capable to express the full scale of natural (at least written) language, including scientific texts, conversations, prose, even poetry.

Multimedia integration

In general, we conclude that the interface we are designing should propose a *multimedia* integration, making it possible to evoke associative meanings in a maximum of different modalities. One may find inspiration about how to express these associative meanings by looking at art (including music, literature, and "applied arts" such as publicity). It is indeed here that the evocation of experiential, subjective and intuitive meanings has been explored most

intensively. It is no accident then that semiotic categories typical of art, such as metaphors and icons, are now being used for the design of computer interfaces.

3.2. EXTERNALIZATION OF ASSOCIATIONS

In addition to evoking associations in the user's mind, one should also find a way to *externalize* the associations, such that they can be explicitly implemented in the computer representation of the problem. *Hypertext* is a paradigm which makes this possible. Hypertext consists of chunks of information (text) which are organized not in the traditional linear sequence which characterizes ordinary text (or computer memory at the machine level), but in a free *network* format. Each chunk can have pointers to an unlimited number of other chunks, designing associations which exist between the chunks (Conklin, 1987; Halasz, 1988). The user is supported in creating, editing and deleting both chunks (text fragments, nodes) and pointers (connections, links).

The integration of multimedia and hypertext interfaces may be called *Hypermedia*. Here too, commercial applications are already easily available. E.g. HYPERCARD on the Apple Macintosh constitutes a quite rich and user-friendly hypermedia environment, which can be used by a non-specialist to develop his or her own computer support system. It enables the combination of text, graphics, sound and animation in a network format where the pointers are implemented as “buttons” which can be clicked with a mouse, thus activating a little program (a “script”) which processes given data or opens to new data.

The problem with HYPERCARD and Hypermedia in general, is that they constitute a purely open support system, which offers much flexibility, but little “intelligence” in helping the user to solve complex problems. The reason is that nothing in the Hypertext concept helps one to decide which chunks and links to introduce, or what meaning to attribute to a specific link. To take an extreme example, a particular association between two chunks might as well be interpreted as “is similar to” or as “is the opposite of”. It is completely up to the user to remember what the link means, or which conclusions to derive from it.

One way to avoid this ambiguity problem consists in introducing constraints which only permit particular *types* of links, with fixed meanings, for example “is an instance of”, “implies” or “causes”. This resembles the approach taken in expert system shells, for example those based on semantic networks. In that case, the system can use the links introduced by the user in order to make inferences and to answer queries. However, the drawback is that the system becomes more like a closed support system, meaning that the input by the user is artificially restricted, and it becomes difficult to spontaneously express the intuitions emerging from associative memory.

A more general way to make a hypermedia network more intelligent will now be sketched.

4. Translating between associative and formal representations

4.1. DOUBLE-LEVEL REPRESENTATIONS

A simple way to combine the advantages of open and closed network representations would consist in combining informal, free, undefined chunks and links with formal, constrained ones. A medium permitting both free, context-dependent expressions and formal expressions may be called a "double level language" (Robinson, 1989). The informal level can be used to express the ambiguous, variable, context-dependent aspects of the problem, and corresponds to what we have called associative memory. The formal level functions as a way of registering the precise, invariant, well-defined features, and corresponds to what we have called the discrete or computer memory. In order to work effectively, the two levels in such a double-level system must have a strong interaction. Let us examine how this can be implemented in a hypermedia system.

On the *informal level*, context-dependency is created by the subjective associations between the information contained in the chunks and what the user(s) have in mind, but which is external to the network itself. For example, a chunk might contain a digitized photograph of a person. This image may have a specific meaning for me, because I know that person, or because that photograph reminds me of someone I know. On the other hand, it may have a completely different connotation for someone else who has different experiences.

On the *formal level*, the system should be able to interpret the mathematical structure formed by the network of chunks and links, independently of the context-dependent information contained within the chunks. The formal meaning of a chunk is purely internal to the network, independent of the user and his or her state of mind, and can only change if the explicit structure of the network is changed.

In the limit of a fully formalized network, the chunks would be empty. Their meaning would be completely determined by their position in the network structure, i.e. by their relations with the whole of the other chunks (cf. Heylighen, 1990c,e). In a completely informal system, on the other hand, there would not be any distinct chunks related by links: the information would be expressed in the form of one continuous "stream of consciousness", consisting of natural language and images.

What we want is to be able to move efficiently between those two extremes during the man-machine conversation, in order to attain that degree of formalism which is most adequate for the problem domain, i.e. that representation which makes optimal use of the problem-solving capacities of both human and computer with respect to the given domain. This means that a relatively well-defined, invariant problem—for example a mathematical or chess problem—would be represented with a high degree of formalization. On the other hand, an ill-defined or "externally structured" (de Zeeuw, 1988) problem—

for example a social or psychological problem—would be represented with a low degree of formalization, and a high degree of external associations. "Formalizing a problem" would then signify: transforming external, variable associations (context-dependent information, metaphors, icons...) into internal, invariant ones (links between nodes).

This is one direction of the digital-associative conversion we want to design. The other direction would translate complex formal structures into concepts meaningful for the user. There are two stages here: 1) specific structures must be recognized and (abstractly) interpreted; 2) the significance or relevance to the problem of these formal interpretations must be presented to the user in a way which is intuitively understandable. Let us now sketch a general methodology for such a two-way translation.

4.2. TOOLS FOR CREATING CHUNKS AND LINKS

Check-lists

The formalization of associative, intuitive knowledge begins with the creation of (multimedia) chunks by the user. The system might already assist the user in this stage by proposing a *check-list*, containing features which might be relevant for the problem domain. These features could be either *universal*, reminding the user of features to be found in any type of problem (for example: which are the relevant objects, properties, operators, constraints, goals, values, ...?, see Heylighen, 1988, 1990b) or *specific*, already containing specialized knowledge about the domain (for example the check-lists used by architects for designing houses with items such as: heating, electricity, bathroom facilities, garbage collection, ...). The function of a check-list is to complement the unreliable recall mechanisms of short and long term memory. Of course, a check-list should never impose anything on the user, it only makes suggestions: if the user does not want heating in his or her house, or does not want to formulate any specific operator, he or she should be free to structure the problem in that way, without being penalised because the support system would get into trouble.

A check-list might be structured as a simple list of items which can be accessed at any point in the network (like a kind of pull-down or pop-up menu). However, it seems better to have a check-list structured in the most general way, namely as a subnetwork. Such a subnetwork could be defined as a "*schema*" (cf. Akscyn et al., 1988): a structure of chunks containing variable parts. Some of the chunks in this structure would already be filled in with typical features, functioning as a template, or frame with "slots", which prompts the user to fill in the empty slots with more specific information. Such schemas can be used to build networks with common parts, simply by copying the schemas and filling in the variable parts manually. The user would be able to edit the schema by deleting irrelevant features or by adding new ones. In fact the user could create new schemas for further use in the same way as he or she would create a new

network. In the end the only real difference between a check-list or schema and an ordinary subnetwork would be that a schema is supposed to contain (more or less) invariant parts which can be applied to many different problem situations.

Outlining

A further facility provided by the system would allow the subdivision of chunks into smaller chunks, or the grouping of chunks so that they form larger chunks. This is what is done by *outliners* or idea processors. Their basic structure is a tree or hierarchy, consisting of a mother chunk with daughter chunks, which themselves are the mothers of daughters, etc., with the constraint that a daughter can only have one mother. This tree can be edited by adding, deleting and moving branches and leaves within the hierarchy.

In order to make this work, chunks should be provided with an *export* and *import* facility, which allows one to select part of their content and export it to a new daughter or sister chunk, or inversely to add to their content (part of) another chunk. This is especially useful if the user does not have any clear distinctions in mind when he or she starts to write down ideas. It would then be possible to first keep everything in one big chunk, as a continous stream of concepts and associations, in order to later segment this chunk into daughter chunks by exporting separate ideas (cf. Halasz, 1988).

String search

The next logical stage of the formalizing process consists in creating links between the chunks. (In practice all the stages should be able to proceed simultaneously, for example new chunks can be introduced after the existing chunks have already been connected). Here too several support mechanisms can be used to attract the attention of the user to interesting candidates for links. On the text level, the chunks can be provided with a *search facility* which allows one to find a certain text string in any of the chunks. The assumption is that a word or word group which is found in two different chunks may point to some sort of association between the two chunks. The user can then judge whether that association is important enough to create a link.

Graphic representation

It is clearly useful to provide the user with a pictorial representation of the network he or she is creating. The chunks can be represented by nodes with a (short) name, and the links by arrows connecting the nodes. Such a "*graphical browser*" can help to alleviate the problem of *disorientation* typical of a hypertext with many nodes. By following links in a complex network, the user can easily get lost, not remembering the path he or she has followed. By looking at a spatial map of the network the user can situate the position of the present node (which may be highlighted) with respect to the other ones. However, when the network is large, it is impossible to show all the nodes on the screen. This problem can be

alleviated by combining the browser structure with the outliner (hierarchic) structure: the browser would only show all the daughter chunks of the present mother chunk. By moving up or down in the hierarchy, the image would zoom out or zoom in, showing more general (mother), respectively more specific (daughter) chunks.

In addition to browsing, the function of a graphic representation would be to help the user to construct and restructure the network. The spatial grouping of the chunks may be a way to express the intuitive "strength of association" which exists between the chunks, such that strongly related chunks would be positioned close to each other on the screen. The user should dispose of elaborate editing facilities for (re)drawing, moving and erasing nodes and arrows (cf. Kommers, 1988). He or she should have the feeling that the nodes representing chunks behave like real objects that can be manipulated (moved, created, destroyed, zoomed in or "opened", zoomed out...).

Matrix representation

Other representations of the network may also trigger important associations in the mind of the user. For example, a network can also be represented as a *matrix* where the rows would for example represent the first or "input" chunk of a link, and the columns the second or "output" part. A matrix listing all chunks (in a particular class, such that the matrix would not become too large) could then be used to create links by marking the elements of the matrix corresponding to important associations. The matrix format is well-known from spreadsheet applications. Its advantage is that all potential links with a given input or output (group of) chunk(s) can be systematically examined by following the corresponding row(s) or column(s).

4.3. CHUNK AND LINK TYPES

It was noted that the imposition of a limited number of chunk or link types "closes" the system, by restricting the spontaneous expression of associations. However, the additional structure generated by such types or classes may still be used, on the condition that types are introduced *freely*. This means: 1) that the user should be able to create new styles or edit existing ones; 2) that there would always be a default, *generic* type, which is neutral in the sense that it does not have any a priori properties or constraints and can be used in any context.

In order to avoid misunderstandings, however, it seems wise to use the convention that a generic link type would design a "positive" association or correlation, meaning that if A is linked to B:

$$A \rightarrow B,$$

then given A, one may expect to see certain features of B, i.e. B would be *more* "probable" or more "actual", and not *less*. If one wants to define the generic link

more sharply, then it would mean : **if** A, **then** *B*. (this does not assume logical implication, characterized by formal properties such as contraposition, it may also denote, for example, the fact that *A* causes *B*, that *A* is a part of *B*, or that *A* refers to *B*). Negative associations (for example, "is inconsistent with", "is the opposite of") could then be expressed by special types of links.

In the case of chunks, the simplest way to implement types seems to be by using the hierarchical ordering of chunks determined by the outliner function. Chunks higher in the hierarchy might then function as more abstract classes or types from which the lower-order chunks would be instances. Such "type" chunks might be proposed in the form of a checklist or schema, as discussed above. The hierarchical relation would then function as an "ISA" relation like in semantic networks, or as an inheritance hierarchy like in object-oriented systems. The idea is that lower order chunks would somehow "inherit" their type (and some of their properties) from their ancestors in the hierarchy.

Potential link types may also be suggested by presenting a checklist. Here too one could distinguish universal link types (for example, "implies", "causes", "is an instance of", "excludes",...) from more specific ones (for example, for family relations: "is the father of", "lives together with", "is older than"...).

However, there is a difference between chunks and links as far as typing is concerned. Indeed, the basic philosophy is that free, context-dependent information is concentrated *within* the chunks, whereas formal, invariant characteristics are expressed through the distinctions and links existing *between* the chunks. Hence we do not want to make links dependent on informal, user-defined labels or types.

A possible way to evade this difficulty would be to format the typed links between chunks as chunks themselves. One would then only need a generic type of links connecting those "relation" chunks to their "object" chunks. For example the relation *A r B*, can be represented as : $A \rightarrow X$, $X \rightarrow B$, where *X* is a chunk representing an instance of the "mother" chunk *r*, which defines the type of the relation. This procedure is equivalent to the mathematical construction used in graph theory for deriving the "total graph" of a given graph *G*. It consists in replacing all nodes and links of *G* by nodes of the total graph *T(G)*. Two nodes (*A* and *X*) in *T(G)* are linked ($A \rightarrow X$) if and only if they correspond to a node and a link in *G*, such that the one is connected to the other (Bakker, 1987).

4.4. FORMAL PROPERTIES AND INFERENCES

The advantage of introducing link types, in addition to the evocation of associations, is that typed links can be characterized by specific *formal properties*, in contrast with generic links which are by definition featureless. Such formal properties determine constraints that can be used by the system to make *inferences*, i.e. to find or to generate (combinations of) chunks which are

connected to the given chunks even though no direct link has been introduced between them by the user.

This may be illustrated by a transitive relation or link type *r*, characterized by:

For all A, B, C: if A r B, and B r C, then A r C.

An example of a transitive link type would be "is the ancestor of", or "implies". Suppose the user has introduced the two links *A r B*, and *B r C*, then implicitly the system will assume that also the link *A r C* has been made, even though the user may not be aware of that link. That may seem trivial, but suppose there is a large set of links, part of which can be ordered in the form of a chain, e.g. *A r B*, *B r C*, *C r ...*, *... r Z*, then it may be quite a surprise for the user to find out that *A r Z*. If starting from *A*, the system infers *Z*, this will appear to the user as an effectively "intelligent" step: something which is (tautologically) true given the constraints introduced, but unexpected from the point of view of the user.

Another example of a formal property relating different types *u, s, t* of links, where *u* might stand for "is the mother of", *s* for "is married to" and *t* for "is the mother-in-law of":

For all A, B, C: if A u B, and B s C, then A t C.

Characteristic for such formal properties is that they are universally valid, for every instance of a type, and hence independent of the context in which the link was introduced, in agreement with our general definition of formal as "having an invariant meaning". More specifically, the kind of properties we are interested in are those which determine the composition or concatenation of links. In the example above, *t* can be defined as the composition of *u* and *s*. In the first example, *r* is equal to the composition of *r* with *r* . An inference can in general be defined as a process in which a concatenation of links is traversed. The result of that concatenation is determined by the formal properties of the links. Reasoning or inference-making in a knowledge network can hence be modelled by the operation of link composition or "link integration" (Bakker, 1987).

These examples show how a simple network supplemented with a few formal properties can behave like a genuine "inference engine" typical of an expert system. The inferences made will of course become much more complex, unexpected and hence "intelligent", if multiple formal properties are introduced, e.g. symmetry, cyclicity, ... (see Heylighen, 1989a,b, for examples of other basic formal properties characterizing networks). Making complex inferences is typically a task in which a computer with its discrete, reliable, and high capacity memory will be much better than a human relying on an associative memory with a low capacity for remembering the "chunks" which determine the different steps of the inference process. This explains why formal problem domains such as

mathematics, physics or engineering appear so difficult to most persons, in contrast to more associative domains like art, literature, and the "humanities".

An inference made by the system can help the user in two ways: it can either directly bring the user closer to the solution of the problem, for example because it answers a "query" by the user, or it can point the user to some peculiarities of the network of associations he or she has introduced. For example, if the user believes that $A\ r\ Z$ is incorrect, then he or she must conclude that either the relation r is not transitive, or that one of the intermediate links was wrongly introduced. This will lead the user to change the representation.

4.5. STRUCTURING KNOWLEDGE BY MEANS OF CLOSURE

The problem with inference structures based on links with special formal properties is that the introduction of such links is not very intuitive: in general the user will not be aware that a certain type of link may be for example transitive or antisymmetric. This difficulty may be tackled by letting the system itself analyse or structure the network in order to find promising candidates for formalization. Such structuring of a network should not only produce link types with special properties, but also higher-order chunks, representing the knowledge in more efficient way. This requires a very special kind of "intelligence": the system should be able to find structure within the chaos of a freely assembled network of chunks and links. A specific set of concepts and algorithms for structuring knowledge networks has been proposed by Bakker (1987; see also Stokman & de Vries, 1988).

We will here discuss a more general—though as present less elaborated—approach based on the concept of *closure*. The fundamentals of this concept have been explained elsewhere (Heylighen 1989a,b; 1990c). Let us just summarize the basic properties. Intuitively, a structure is called "closed" if from the outside it can be distinguished as a whole, separated from its surroundings, but none of its internal structures can be distinguished: they are hidden from the outside observer. The closure of a subnetwork within a larger network means that there is an invariant distinction between the chunks and links *within* the subnetwork and those *outside* it, whereas there is no (or a less) invariant distinction between the elements within when considered from the outside. In other words, closure enhances external distinctions, and diminishes internal distinctions.

Chunk identification

The simplest example of closure in a network occurs when two or more chunks are *equivalent* or indistinguishable. This is the case where there are no internal distinctions within the set of equivalent chunks: there is only an outside distinction separating the equivalence class from the other chunks that are not equivalent to them. The system may then propose to the user to merge the

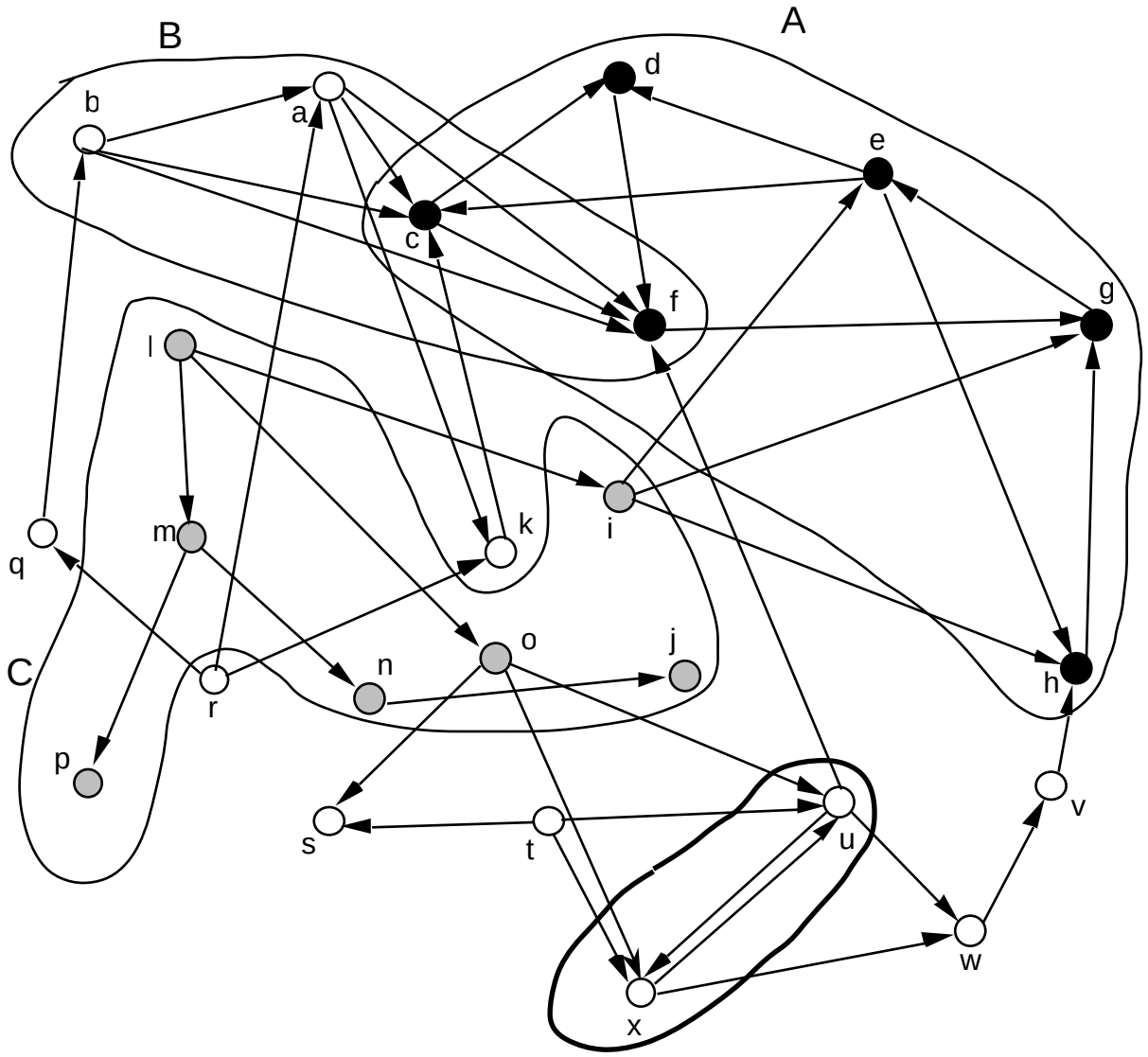


FIGURE 1: a complex network with four closed subnetworks:
{u, x} are equivalent chunks that may be identified. A, B, C can be considered as "higher-order" chunks integrating their "lower-order" chunks: A = {c, d, e, f, g, h} (black nodes) is an attractor; B = {a, b, c, f} (black and white nodes} is a transitively closed network; C = {p, l, m, n, o, i, j} (grey nodes) is a network in the form of a tree. The other (white) nodes do not belong to any closed subnetwork. Remark that the nodes c and f belong to two different closed subnetworks, A and B.

separate chunks, i.e. to consider them as identical, and to replace them by a single chunk.

Chunks can be assumed to be equivalent within the network when they have exactly the same links with the whole of all the other chunks (see fig. 1). In that case, they are indistinguishable by means of these links alone: they play the same role in the network, and hence may be identified (according to the axiom of the identity of the indistinguishables, cf. Heylighen, 1990c,e). (A weaker example of this same property, discussed by Bakker (1987), occurs when the links of one chunk form a subset of the links of another chunk.)

A specific case of this situation occurs when the equivalent chunks are mutually connected by links, but such that every chunk of the class is directly connected to every other chunk of the class. This means that the links within the class form an equivalence relation (characterized by symmetry and transitivity). Another example of this situation occurs with chunks that have no links with other chunks at all: two empty sets of links can be assumed to be identical.

Chunk integration

An equivalence relation between chunks is an example of a perfect closure, where all chunks internal to the equivalence class behave indistinguishably from any viewpoint. However, one can also consider different types of "partial" closures, where the internal elements of the closed subnetwork are perhaps not completely equivalent, but where their distinction is clearly less important, i.e. less invariant, than the distinction between internal and external.

In that case it becomes meaningful to introduce a "higher-order" chunk (called "frame" in Bakker, 1987) representing the closed subnetwork in a more abstract manner (fig. 1). The chunks internal to the subnetwork are not identified or merged, like in the previous case, they are "integrated". This means that they still retain their local identity (internal distinctions), but that they are no longer distinguished from a more global, less detailed point of view. The higher-order chunk is connected to the lower-order chunks it represents by a hierarchical (mother-daughter) link.

An example of a "partially" closed subnetwork is an *attractor*. An attractor is defined as a subnetwork such that no link ("arrow") leaves the subnetwork—there are only incoming arrows—, and such that no part of the subnetwork constitutes an attractor itself (fig. 1). This means that any path of arrows leading from a chunk *a* to a chunk *z* in the attractor must have an inverse path, leading back from the final chunk *z* to the initial chunk *a*. Otherwise, *a* could no longer be reached from *z*, and *z* would function as an attractor with respect to *a*, in contradiction with the requirement that there be no subattractors. Such a subnetwork may be called "*cyclically*" closed: every path has an inverse path, together forming a cycle. Intuitively, an attractor determines a boundary (the external distinction) that can be transgressed only in one direction: one can get in, but one cannot get out, and once one is in, one continues to cycle among the elements of the attractor.

With the if-then interpretation of links, an attractor is clearly an interesting formal knowledge structure: if one is in the attractor, one remains within it. Given the attractor, one does not need to check all possible links in order to reach this conclusion. This is another example of a non-trivial inference the system can make on the basis of the formal properties of an attractor relation. An attractor could be interpreted as a kind of local optimum, e.g. a "minimum of the potential".

Complementary to a cyclically closed network is a *transitively* closed one (fig. 1), which we already defined in the paragraph on inference structures. The relative lack of internal distinction between chunks connected through a transitive relation is exemplified by the fact that all chunks indirectly connected through a path, are also directly connected through simple arrows. Cyclical and transitive closure together define equivalence closure: all chunks in the subnetwork are linked to all other chunks.

Another important closure type is called *surjective*: every chunk in the connected subnetwork has at most 1 incoming link, so that the subnetwork forms a tree, see fig. 1. There are further a number of different, but related closures (cf. Heylighen, 1989a,b; 1990c), such as horismotical, orthogonal, parallel closure..., which can be combined in more complex closures. Each is characterized by a specific formal structure, leading to a specific type of distinctions, enabling the restructuration of the network by introducing higher-order chunks representing the closed subnetwork, with a specific cognitive interpretation. Remark that in certain cases different types of closures can overlap (see fig. 1): the same chunk may belong to two higher-order chunks of a different type. In order to model this within the Hypermedia network, the hierarchical (tree) ordering must be generalized to a partial ordering ("multiple inheritance").

The formalism and interpretation of all these closure operations must be further elaborated, but the fundamental idea, which consists in finding invariant distinctions, seems clear. It proposes a very promising method of translating complex, disordered associative knowledge into relatively simple, ordered, formalized knowledge, thus replacing densely connected networks of associations by a few higher-order chunks, connected by formally defined relations.

Complexity reduction vs. knowledge elicitation

The replacement of a set of chunks by a single chunk is an example of a *complexity reduction* of the knowledge network, simplifying the search for solutions within the network. It may be useful here to remark that "solving" a problem is just a limit case of restructuring or "reformulating" a problem (cf. Simon, 1981): the solution is that reformulation where the problem or search space (of potential solutions) has been reduced to a single element. In general the objective of a support system for problem-solving could be defined as: making the search space as small as possible. The system is not required to find the final solution: if the problem space has been sufficiently simplified and if the consequences of the different alternatives have been made sufficiently clear, the user can in general make the final decision as to what solution he or she should choose.

However, when the system analyses the network, finds a closed subnetwork and proposes to identify or integrate the chunks of that subnetwork, the user may disagree with the proposed reduction. The user may think that these chunks are not identical (or not even similar), because they have different associations with

the context, i.e. with things he or she has in mind, but which were not introduced in the network. In that case, the system will suggest to add these external associations to the network, in order to make the implicit distinction explicit, by adding links or chunks, which disambiguate the apparent closure. This is an example of *knowledge elicitation*, where the user is prompted to make explicit his or her intuitive knowledge, thus adding more variety to the network. Instead of being distinct by their label or content (i.e. by their external associations), the chunks will now be distinguished by their position in the formal structure, i.e. by their links internal to the network.

A good interactive support system should stimulate the user to question apparent closures, possibly resulting in the "opening up" of an inappropriately closed network. A traditional example of premature closure is the problem where one is given a number of dots, arranged in a square matrix. One is then asked to draw a maximum number of straight lines such that each dot is on just one line. Intuitively, one assumes that the square forms a closed region which one should not leave while drawing the lines. However, the problem is structured in such a way that one can only solve it by leaving the square, i.e. by "breaking up" the closed "Gestalt".

Complexity reduction and knowledge elicitation are two complementary results of the structuring of a network by looking for closure. The former reduces variety and hence the needed search for a solution within a given problem formulation; the latter increases variety by attracting attention to additional resources or constraints which are relevant to the problem but which are as yet not included in the problem formulation.

5. CONCEPTORGANIZER: a prototype of an interactive interface

In order to experiment with and to test the basic concepts expressed in this paper, a first prototype (the "CONCEPTORGANIZER") of an interactive Hypermedia system supporting formal-associative conversions has been implemented. Though the system is still quite simple, and basic features such as the search for closure are incompletely formalized, it can be used to illustrate how some of the design ideas put forth may be realized in practice.

The basic medium in which the CONCEPTORGANIZER is programmed is the Hypermedia development environment HYPERCARD, with its incorporated programming language HYPERTALK. Though the language is very easy to use, it offers most features of an advanced software development system, including features similar to imperative, object-oriented and functional programming. The advantage of using HYPERCARD is that one gets a user-friendly, multimedia, hypertext environment for free. What must be added to it is some form of intelligent support of the user, helping to express and reorganize his or her knowledge in order to make it better suited for problem-solving. This requires

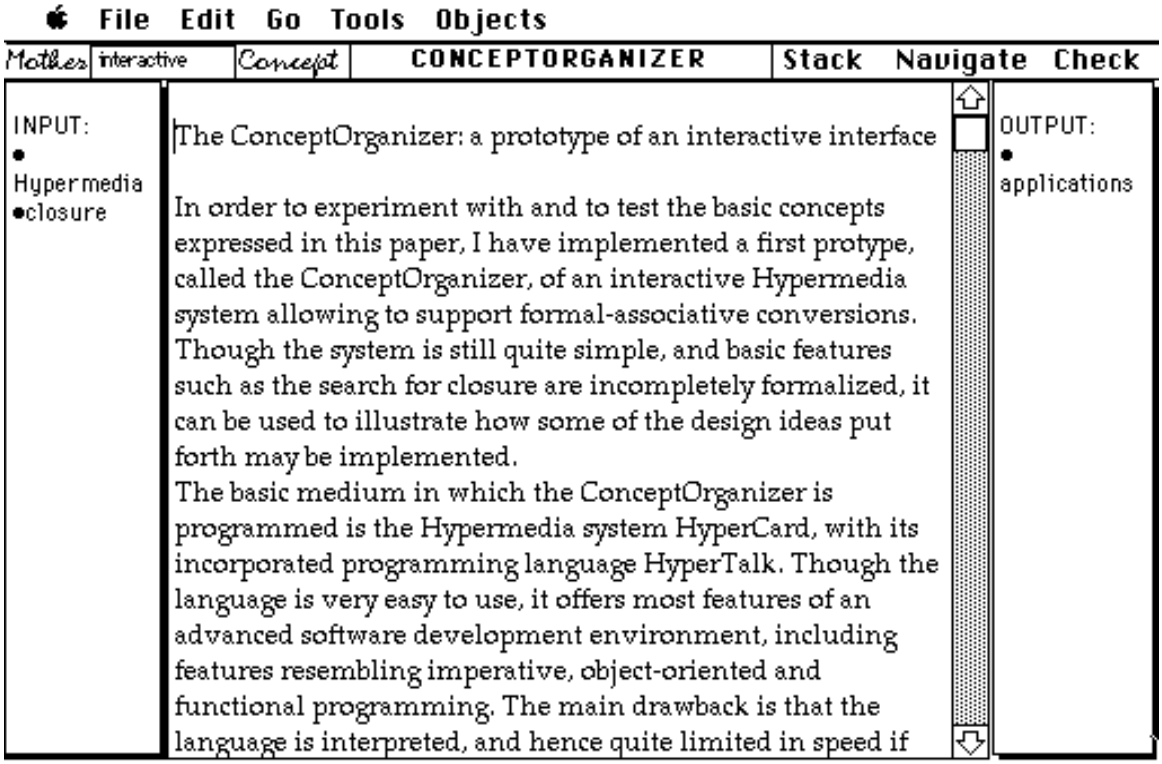


FIGURE 2: a card representing a concept in text mode.
The upper row ("File", "Edit", ...) contains the HYPERCARD pull-down menus, the second row contains the name of the Mother concept, the name of the concept, and three CONCEPTORGANIZER pull-down menus ("Stack", "Navigate" and "Check"). The left text field contains the names of the input concepts, the center (scrolling) field contains the description of the concept, and the right field contains the names of the output concepts.

routines for: 1) easily representing, manipulating and editing chunks and their links; 2) analysing networks of chunks in order to find closures; 3) supporting an inference engine, which allows one to intelligently query the network.

Chunks (which are called "concepts" here) are represented basically by HYPERCARD "cards", which may contain text fields, graphics, and buttons. The text describing the chunk will be contained in the main field of the card (see fig. 2). Links to other chunks (cards) are expressed by an input field, containing the names of chunks which are linked to the present chunk, and an output field containing the names of chunks to which this chunk is linked.

CONCEPTORGANIZER and HYPERCARD commands can be accessed through pull-down menus. Apart from the "horizontal", associative structure, the network also contains a "vertical", hierarchical (tree) structure, such that each chunk has a "mother" chunk. The network can be navigated easily both in the horizontal direction, by selecting (with the mouse) the name of the input or output chunk one wants to go to, and in the vertical direction, by clicking the mother button, or by clicking one of the (several) daughter buttons.

The daughter chunks are represented by node-like buttons which become visible when the text fields are hidden (see fig. 3). This is the graphical

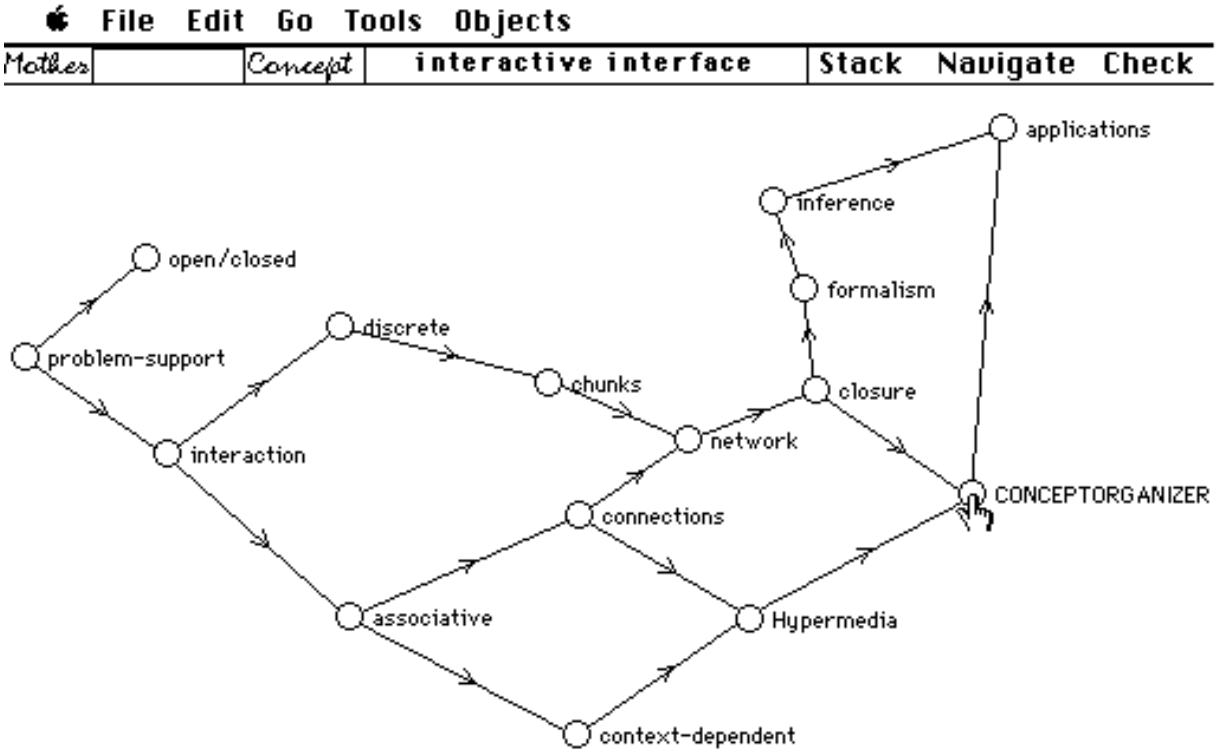


FIGURE 3. A card representing the mother concept of the previous concept in graphic mode. The previous concept can be reopened by double-clicking on its node representation (on the extreme right, in the middle).

representation mode of the network. In this mode, new chunks are created by simply clicking on the empty background at the position where the new node should come. The system then puts up a dialogue asking for the name of the chunk. If there is already a chunk with this name, the user is warned. If the new name is acknowledged (OK button), a new card is created linked to the present card via the mother button. The node which graphically represents the new chunk may be linked to one of the other nodes on the original card simply by clicking on the first node (which is then high-lighted), then clicking on the second node. This results in an arrow being drawn between the two nodes, and in the names of the linked chunks being added to the appropriate fields on their corresponding cards. In order to go to one of the daughter cards represented by the nodes, it suffices to double-click on the appropriate node and the card is "opened", showing the text defining the chunk and the in- and output fields. The graphical network can be edited by creating, deleting and moving nodes, and by erasing and redrawing their links.

It is of course also possible to create links without using the graphical mode, for example by selecting a word in a chunk text and letting the system find other chunks which contain the same word(s). The system then asks whether it should connect both cards. New chunks can be created from existing ones by selecting part of the text, and choosing the "export" command which copies or cuts the text

from the present card, creates a new daughter card, and puts the text in its text field. This is useful when an existing (long) text (e.g. a paper) is used as the substratum for creating a network. The text is simply imported in one (big) chunk, which is then step-by-step split up into smaller daughter chunks, which are connected to each other in the second phase.

As yet the analysis of closed subnetworks is implemented only partially. It may be exemplified by a command checking whether there are two chunks which have the same links (i.e. identical sets of input and output chunks). If such chunks exist, the user is attended to the resulting ambiguity, and suggested to resolve it by either merging the chunks, or introducing additional distinguishing links. Another command searches for attractors, by considering the whole of links as defining a function mapping sets of "input" chunks onto sets of "output" chunks. A set of chunks then contains an attractor if it is equal to its image through this function. By starting from minimal sets (singletons) and expanding them, one is certain that no subattractors are contained within the set, and hence that the set is a genuine attractor. Similar commands are being developed for checking the existence of repulsors (inverse attractors) and transitive subnetworks. Once an attractor (or otherwise closed subnetwork) is found, a new mother chunk may be created representing this attractor.

Until now the only way of querying the system consists in searching for chunks with specific names or containing specific word combinations, or in following the horizontal or vertical links in the direction one is interested in. A somewhat more complex example of querying a HYPERCARD knowledge representation system, which uses several types of links (e.g. "Is A", "Causes", "Sign Of",...), but no real formal properties, can be found in (Bonadona, 1989). However, a more powerful inference engine is being developed on the basis of an adaptation of the principle of resolution (which is at the base of knowledge representation systems using predicate logic, cf. Jackson, 1986) to the network format and the closure principle.

6. Applications

6.1. GENERAL OBJECTIVES

Let us us sketch some of the practical domains to which a system as proposed here might be applied. The overall objective of the system may be summarized as follows: to elicit from the user(s) all the knowledge which is relevant to the problem domain, to structure that knowledge in the most transparent and simple way, and to use the knowledge network built up in this way in order to support problem-solving.

On the level of problem-solving, if the problem is well-structured and if all the relevant knowledge is formalized, the knowledge network would function as an *knowledge based system*, which directly answers queries. The difference with a

traditional expert system would be that the system is still open, i.e. it is easy to introduce new knowledge or to change the existing knowledge, thus reformulating the problem domain. Complementary to the expert system interpretation of the network (characterized by backward-chaining, goal-driven querying), is the use of the network as a *qualitative simulation* of the domain (characterized by forward-chaining, data-driven inferences, cf. Hornung, 1982, 1987).

In the general case, however, the knowledge would not be completely formalized, and hence the problem must be solved by user and system in interaction. Assuming that all formalizable knowledge has already been introduced, the function of the system would be to maximally simplify the problem representation, i.e. the space of potential solutions, and to clearly map out the implications of each of the potential choices. The task of the user would consist in making the final decision, based on his or her own intuitive knowledge together with an assessment of the implications shown by the system. The system would hence function basically as an interactive *decision support*.

The problem-solving process as we have sketched it is not restricted to an individual user. The in-built flexibility of the interface allows knowledge to be introduced by different users, and the structuring mechanisms allow the system to integrate (or show the inconsistencies between) ideas from very different origins. Hence the system can also be applied to *computer-supported co-operative work* (CSCW), for example as an intelligent aid to brain-storming or to group decision-making.

6.2. SIMILARITIES AND DIFFERENCES WITH EXISTING APPLICATIONS

Existing Hypermedia applications can be classified according to whether they put the emphasis on *browsing* or on *authoring*. In the first type of systems the objective is to make it easy to find information in a large and ill-structured database, for example an encyclopedia, a user manual, or a library of linked documents. The user is supported in browsing through the stored data, but not in configuring the network. The second type of hypermedia systems are primarily meant as supports for “idea processing”, that is to say (re)organizing loosely structured data, entered by one or several users, for example in order to design a system, plan a project, outline a document, or reach consensus and collaborate on a collective issue. The system which is proposed here belongs clearly to this second type of applications.

Though one might distinguish between authoring systems aimed primarily at a single user (e.g. NoteCards, Halasz, 1988), and systems emphasizing group communication and cooperation (e.g. ZOG/KMS, Akscyn et al., 1988; gIBIS, Begeman & Conklin, 1988), there does not seem to be any fundamental difference in the way data are organized in these two classes of applications. In both cases, unconnected or loosely connected data (ideas, viewpoints, facts, issues, concepts...) are introduced more or less independently of each other. It does not make much

difference whether they were introduced by the same or by different users: the main point is to structure them in a meaningful, transparent and consistent way. One can just expect that the more diverse the users are, the less obvious the connections between their ideas will be.

Such problem structuring hypermedia systems use different ways of creating networks. Most of them incorporate different types of chunks or links. Sometimes the types are rather specific (for example, in gIBIS, a chunk must either be an “issue”, a “position”, or an “argument”), sometimes they are very general (for example, in ZOG/KMS, there are two basic types of links, like in CONCEPTORGANIZER: hierarchical relations, and a generic “reference” type). In NoteCards, new types can be defined by the user. Apart from that, the main differences between those systems reside in the user interface: the presence or absence of different kinds of graphical browsers, the possibility of having multiple windows, the use of mouse and menus, etc.

What distinguishes the Hypermedia system proposed here from those systems is that the latter systems are still merely “open.” The problem structuring is supported in a purely passive way, by providing editing facilities, chunk and link types, etc. The system itself will never propose a particular way of structuring the input. Such systems are confronted with a lot of difficulties if the domain becomes really complex. On the basis of his experience with NoteCards and its weaknesses, Halasz (1988) has listed a number of issues which should be addressed by the “next generation of Hypermedia systems”. It can be noted that a number of his proposals would be (explicitly or implicitly) adopted by the system proposed here, for example: the search for structure in addition to search for content, the possibility for the system to reconfigure itself, the integration of knowledge-based systems, the extensibility and tailorability of the system by non-expert users.

On the side of the “closed” support systems, it must be noted that many recent expert systems are beginning to use ideas from Hypermedia. However, the Hypermedia organization is in general limited to the user interface, helping the user to find information, definitions or explanations in a browsing manner, but without support for authoring new knowledge (cf. Bonadonna, 1989).

7. Conclusion

It has been argued that an intelligent and flexible support system applicable to complex problem domains should neither be “open”, i.e. characterized by an active user and a passive system, nor “closed”, i.e. characterized by an active system and a passive user. The system should rather be “interactive”, i.e. characterized by a continuing conversation between user and system which are both active carrying out those tasks each of them is good at, thus complementing each other's weaknesses. In order to design such a system, it was necessary to

understand just which are the kind of problems a human, respectively a computer, is good at.

It was concluded that man is characterized by an associative, context-dependent memory, which is very good at learning complex and ill-structured associations, and at rapidly recognizing aspects of a problem similar—though in general not identical—to aspects experienced before. On the other hand, man is bad at memorizing and recalling simple chunks of information, and thus ill-prepared to tackle problems where many different combinations of such chunks are to be examined. A computer, on the contrary, has no difficulty storing, retrieving and combining chunks, but it is helpless if the problem is not sufficiently well-structured, i.e. if no discrete chunks are given beforehand.

In order to let man and machine interact in the most fruitful way, one needs a "double level" interface that can evoke the context-dependent associations of the human as well as provide a discrete formalism which can be efficiently manipulated by the computer. The context-dependent level consists of a multimedia environment, where the user can freely express his or her intuitive ideas in natural language, by drawings, or by other modalities. The formal level consists of a hypertext network of chunks, which can be structured by introducing specific, formally defined relations, supporting non-trivial inferences.

The conversion from the informal to the formal "language" is based on externalizing subjective associations by introducing explicit chunks and links between chunks, and on defining higher-order chunks (which may represent chunk or link *types*) by looking for invariant distinctions. The first process can be supported by several tools, such as check-lists, an outliner, search facilities and graphical and matrix representations of the network. The second process—finding higher-order invariant distinctions, which introduce some "order" in the network and thus enable complex inferences—can be supported by looking for closed subnetworks.

Closure is a basic concept for structuring or organizing associative networks, because it allows one to "highlight" the more invariant, and hence more important, distinctions which exist between chunks or between (sub)networks of chunks. It can be used to reduce complexity, by replacing a set of irrelevant distinctions by one higher-order distinction. It can also be used to elicit knowledge, by prompting the user to introduce additional chunks and links, allowing the system to distinguish otherwise ambiguous chunks. Inferences can be thought of as search processes through the network of explicit or implicit links, guided by formal constraints, such as transitivity, that can be reduced to some form of closure.

We have not paid as much attention to the conversion from formal to informal representations, because it seems more difficult to formulate general rules here, since each user is different and has different associations. Suffice it to say that the meaning and implications of the different types of closure should be defined and illustrated in the most understandable and intuitive way possible, in

order that the user be able to make decisions on the basis of a real insight in what is happening in the formalism.

The implementation of an interactive interface based on the principles above has been illustrated by considering a prototype application: the CONCEPTORGANIZER. Though many of the main concepts have a counterpart in this system, there is as yet insufficient integration of the different features, so that it is difficult to test the usefulness of the system in tackling really complex problems. It is clear in which direction future research must proceed: development of a flexible and efficient inference engine, integration of formal inference relations and mechanisms for finding closure in the associative network, better support of the user in interpreting and using these formal properties, testing the system in real-world problem situations in order to find and correct the remaining shortcomings.

References

Akscyn R.M., McCracken D.L. & Yoder E.A. (1988): "KMS: a distributed hypermedia system for managing knowledge in organizations", Communications of the ACM 31:7, p. 820-835.

Bakker R.R. (1987): Knowledge Graphs: representation and structuring of scientific knowledge, (Ph.D. Thesis, Dep. of Applied Mathematics, University of Twente, Netherlands).

Begeman M.L. & Conklin J. (1988): "The Right Tool for the Job", BYTE Oct. 1988, p. 255-266.

Beyls P. (1986): "Intimate Machine Interaction; an illustrated definition", The Visual Computer 2, p. 152-158.

Bonadonna F. (1989): "HYPERSHELL: an Expert System Shell in HyperMedia Environment", Wheels for the Mind Europe 2, p. 33-40.

Conklin J. (1987): "Hypertext: a Survey and Introduction", IEEE Computer 20:9, p. 17-41.

de Bono E. (1969) : The Mechanism of Mind, (Penguin, Harmondsworth).

de Zeeuw G. (1989): "Systems Profile - The Experience of Change", Systems Research 6, p. 65-73.

Ditlea S. (1989): "Another World, Inside Artificial Reality", PC/Computing, Nov. 1989, p. 91-101.

Halasz F. (1988): "Reflections on NoteCards: seven issues for the next generation of hypermedia systems", Communications of the ACM 31:7, p. 836-852.

Heylighen F. (1988) : "Formulating the Problem of Problem-Formulation", in: Cybernetics and Systems '88, Trappl R. (ed.), (Kluwer, Dordrecht), p. 949-957

Heylighen F. (1989a) : "Coping with Complexity: concepts and principles for a support system", in: Preceedings of the Int. Conference "Support, Society and Culture: Mutual Uses of Cybernetics and Science", Glanville R. & de Zeeuw G. (ed.), (OOC, University of Amsterdam), p. 26-41.

Heylighen F. (1989b) : "Self-Organization, Emergence and the Architecture of Complexity", in: Proc. European Congress on System Science, (AFCET, Paris), p. 23-32.

Heylighen F. (1990a) : "Autonomy and Cognition as the Maintenance and Processing of Distinctions", in: Heylighen, Rosseel & Demeyere (eds.), p. 89-106.

Heylighen F. (1990b) : Representation and Change. A Metarepresentational Framework for the Foundations of Physical and Cognitive Science, (Communication & Cognition, Gent).

Heylighen F. (1990c) : "Relational Closure: a mathematical concept for distinction-making and complexity analysis", in: Cybernetics and Systems '90, R. Trappl (ed.), (World Science Publishers, Singapore), p. 335-342.

Heylighen F. (1990d): "Non-Rational Cognitive Processes as Changes of Distinctions", Communication & Cognition 23: 2.

Heylighen F. (1990e): "A Structural Language for the Foundations of Physics", Int. J. Gen. Systems (in press).

Heylighen F., Rosseel E. & Demeyere F. (eds.) (1990) : Self-Steering and Cognition in Complex Systems. Toward a New Cybernetics, (Gordon and Breach, New York).

Hornung B.R. (1982) : “Qualitative Systems Analysis as a Tool for Development Studies”, in: Dependence and Inequality, Geyer R.F. & Van der Zouwen J. (eds.), (Pergamon, Oxford), p. 187-219.

Hornung B.R. (1987) : “Expert Knowledge Based Simulation for Health Planning in the Third World”, in: Medical Informatics Europe '87. Proc. of the 7th Int. Congress, Serio A., O'Moore R., Tardini A. & Roger F.H. (eds.), (European Federation for Medical Informatics, Rome), p. 1349-1354.

Jackson P. (1986): Introduction to Expert Systems (Addison-Wesley, Wokingham, England).

Kommers P.A.M. (1988) : “Textvision: Elicitation and Acquisition of Conceptual Knowledge by Graphic Representation and Multiwindowing”, in: Human-Computer Interaction, G. Van der Veer & G. Mulder (ed.), (Springer Verlag, Berlin), p. 237-249

Maurer R. & Schreter Z. (1990): “Sensory-Motor Spatial Learning in Artificial Connectionist Organisms”, in: Heylighen, Rosseel & Demeyere (1990), p. 160-182.

Miller G.A. (1956): “The Magical Number Seven Plus or Minus Two: some limits on our capacity for processing information”, Psychological Review 63, p. 81-97.

Pask G. & Gregory D. (1986) : “Conversational Systems”, in: Human Productivity Enhancement (vol. II), Zeidner R. J. (ed.), (Praeger, New York).

Pask G. (1980): “Developments in Conversation Theory—Part 1”, Int. J. Man-Machine Studies 13, p. 357-411.

Rumelhart D.E. & McClelland J.L. (eds.) (1986) : Parallel Distributed Processing : Explorations in the Microstructure of Cognition (Bradford Books / MIT Press, Cambridge, MA).

Simon H.A. (1957) : Models of Man : Social and Rational, (Wiley, London).

Simon H.A. (1981) : The Sciences of the Artificial, (MIT Press, Cambridge MA).

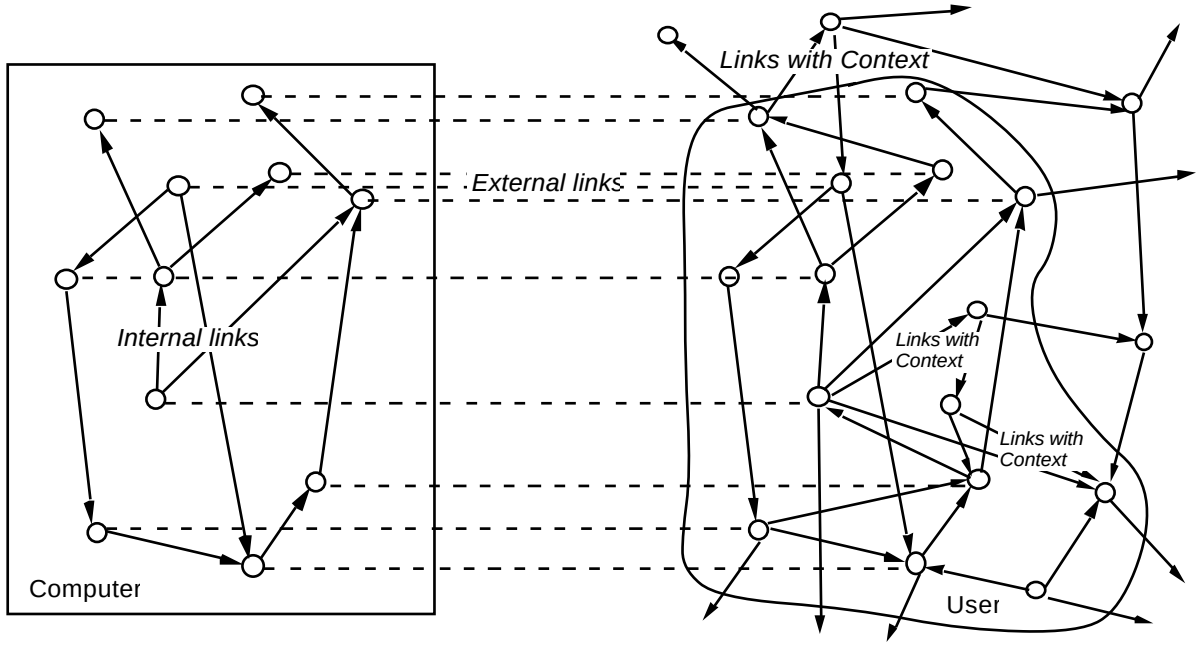
Stokman F.N. & de Vries P.H. (1988): “Structuring Knowledge in a Graph”, in: Human-Computer Interaction, Psychonomic Aspects, G.C. van der Veer & G.J. Mulder (eds.), (“Springer, Berlin).

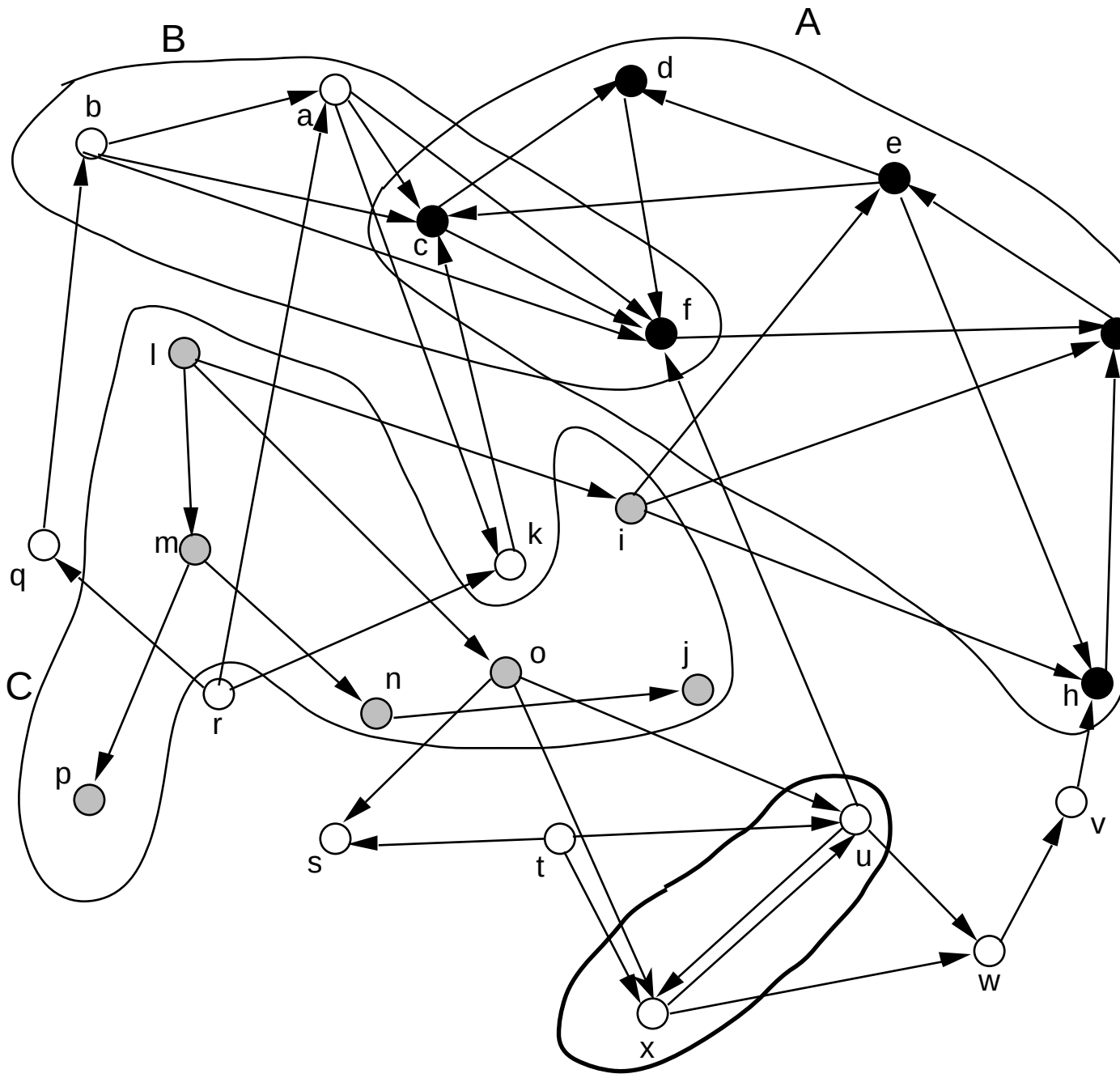
Warfield J. (1988): “The Magical Number 3 Plus or Minus Zero”, Cybernetics and Systems 19, p. 339-358.

Wickelgren W.A. (1977) : Learning and Memory, (Prentice-Hall, Englewood Cliffs, N.J.).

Williams T. (1988): “Input technologies extend the scope of user involvement”, Computer Design 27 (5), p. 41-50.

FIGURE 1: Formal and context-dependent associations.
The Hypermedia network in the computer (left) represents a (small) part of of the network of associations of the user (right). The chunks on the right which are absent in the formal representation one the left correspond to the "context", which may be internal to the user or dependent upon outside phenomena (other people, the environment).




A
B
C
a
b
c
d
e
f
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x

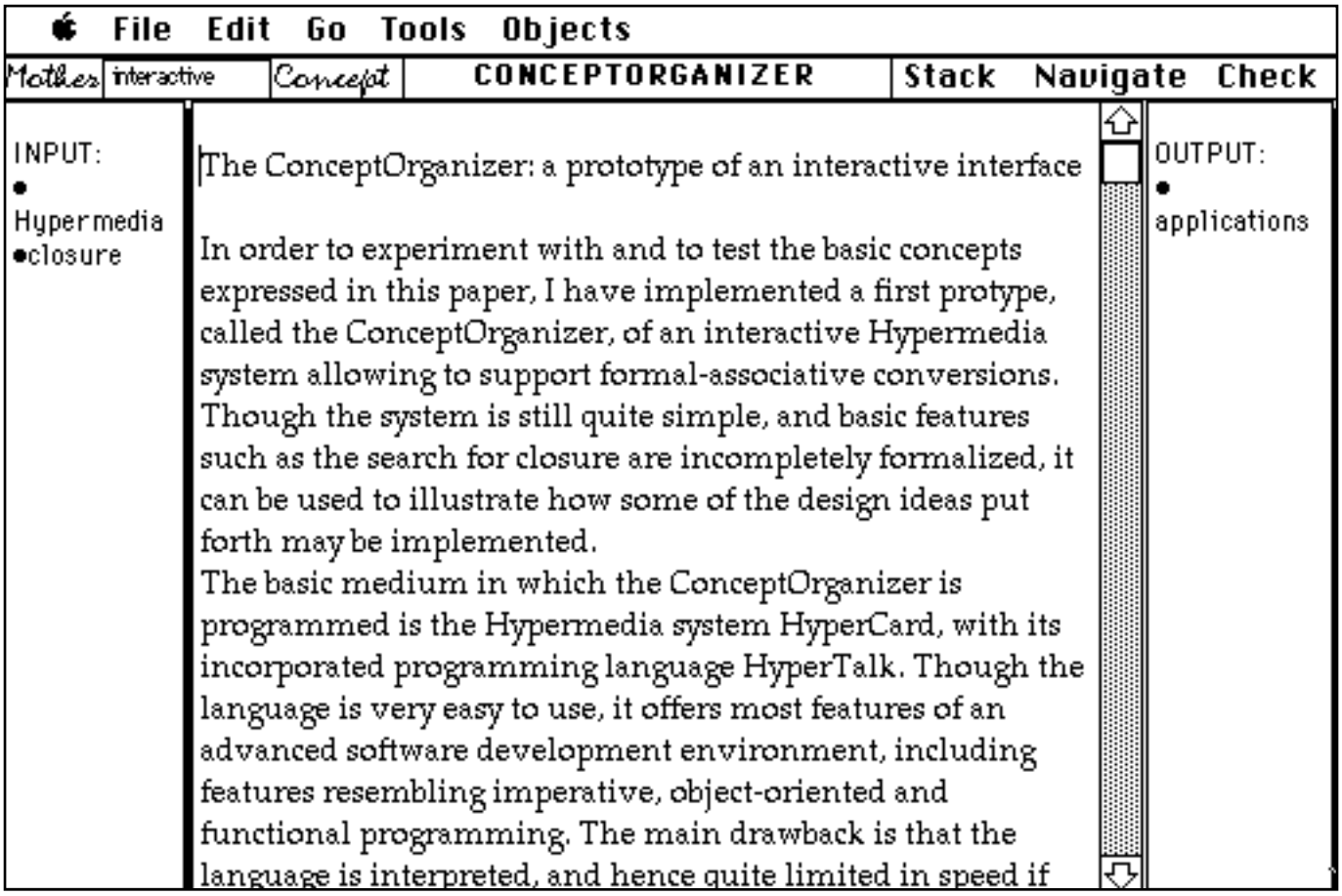

File Edit Go Tools Objects
Mother
interactive
Concept
CONCEPTORGANIZER
Stack Navigate Check
INPUT:
•
Hypermedia
•closure
The ConceptOrganizer: a prototype of an interactive interface
In order to experiment with and to test the basic concepts expressed in this paper, I have implemented a first protype, called the ConceptOrganizer, of an interactive Hypermedia system allowing to support formal-associative conversions. Though the system is still quite simple, and basic features such as the search for closure are incompletely formalized, it can be used to illustrate how some of the design ideas put forth may be implemented.
The basic medium in which the ConceptOrganizer is programmed is the Hypermedia system HyperCard, with its incorporated programming language HyperTalk. Though the language is very easy to use, it offers most features of an advanced software development environment, including features resembling imperative, object-oriented and functional programming. The main drawback is that the language is interpreted, and hence quite limited in speed if
OUTPUT:
•
applications

Fig. 1

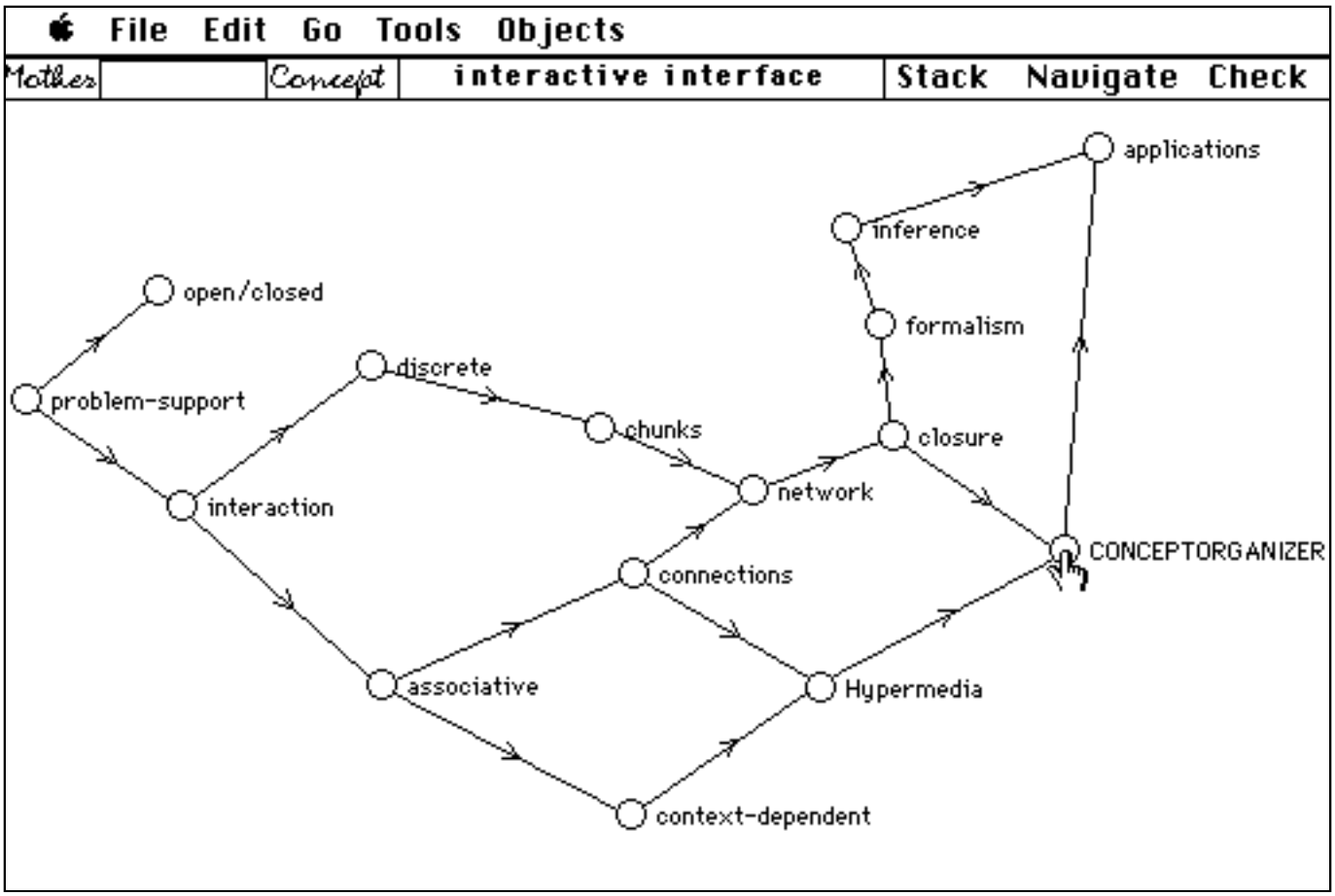

File Edit Go Tools Objects
Mother
Concept
interactive interface
Stack Navigate Check
applications
inference
open/closed
formalism
discrete
problem-support
chunks
closure
network
interaction
CONCEPTORGANIZER
connections
associative
Hypermedia
context-dependent

Fig. 2